Underapproximative verification of stochastic LTI systems using Fourier transform and convex optimization

This example will demonstrate the use of SReachTools in verification and controller synthesis for stochastic continuous-state discrete-time linear time-invariant (LTI) systems.

Specifically, we will discuss the terminal hitting-time stochastic reach-avoid problem, where we are provided with a stochastic system model, a safe set to stay within, and a target set to reach at a specified time, and we will use SReachTools to solve the following problems:

1. **Verification problem from an initial state:** Compute an underapproximation of the maximum attainable reach-avoid probability given an initial state,
2. **Controller synthesis problem:** Synthesize a controller to achieve this probability, and
3. **Verification problem:** Compute a polytopic underapproximation of all the initial states from which the system can be driven to meet a predefined probabilistic safety threshold.

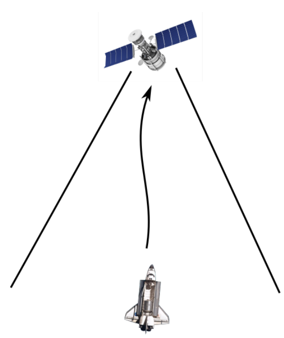
Our approach uses Fourier transforms, convex optimization, and gradient-free optimization techniques to compute a scalable underapproximation to the terminal hitting-time stochastic reach-avoid problem.

Notes about this Live Script:

1. **MATLAB dependencies**: This Live Script uses MATLAB's Global Optimization Toolbox, and Statistics and Machine Learning Toolbox.
2. **External dependencies**: This Live Script uses Multi-Parameteric Toolbox (MPT) and CVX.
3. We will also Genz's algorithm (included in helperFunctions of SReachTools) to evaluate integrals of a Gaussian density over a polytope.
4. Make sure that `srtinit` is run before running this script.

Problem formulation: spacecraft rendezvous and docking problem

We consider both the spacecrafts, referred to as the deputy spacecraft and the chief spacecraft, to be in the same circular orbit. **We desire that the deputy reaches the chief at a specified time (the control time horizon) while remaining in a line-of-sight cone.** To account for the modeling uncertainties and unmodeled disturbance forces, we will use a stochastic model to describe the relative dynamics of the deputy satellite with respect to the chief satellite.



Dynamics model for the deputy relative to the chief spacecraft

The relative planar dynamics of the deputy with respect to the chief are described by the Clohessy-Wiltshire-Hill (CWH) equations. Specifically, we have a LTI system describing the relative dynamics and it is perturbed by a low-stochasticity Gaussian disturbance to account for unmodelled phenomena and disturbance forces. We will set the thrust levels permitted to be within a origin-centered box of side 0.2.

```
umax=0.1;
mean_disturbance = zeros(4,1);
covariance_disturbance = diag([1e-4, 1e-4, 5e-8, 5e-8]);
% Define the CWH (planar) dynamics of the deputy spacecraft relative to the chief spacecraft a
sys = getCwhLtiSystem(4,...
                      Polyhedron('lb', -umax*ones(2,1),...
                                 'ub',  umax*ones(2,1)),...
                      StochasticDisturbance('Gaussian',...
                                            mean_disturbance,...
                                            covariance_disturbance));
```

Target set and safe set creation

For the formulation of the terminal hitting-time stochastic reach-avoid problem,

- **the safe set** is the line-of-sight (LoS) cone is the region where accurate sensing of the deputy is possible (set to avoid is outside of this LoS cone), and
- **the target set** is a small box around the origin which needs to be reached (the chief is at the origin in the relative frame).

```
time_horizon=5;                                                  % Stay within a line of sight cone
                                                                 % reach the target at t=5% Safe Se
%% Safe set definition --- LoS cone |x|<=y and y\in[0,ymax] and |vx|<=vxmax and |vy|<=vymax
ymax=2;
vxmax=0.5;
vymax=0.5;
A_safe_set = [1, 1, 0, 0;
             -1, 1, 0, 0;
              0, -1, 0, 0;
              0, 0, 1,0;
              0, 0,-1,0;
              0, 0, 0,1;
              0, 0, 0,-1];
b_safe_set = [0;
              0;
              ymax;
              vxmax;
              vxmax;
              vymax;
              vymax];
safe_set = Polyhedron(A_safe_set, b_safe_set);
%% Target set --- Box [-0.1,0.1]x[-0.1,0]x[-0.01,0.01]x[-0.01,0.01]
target_set = Polyhedron('lb', [-0.1; -0.1; -0.01; -0.01],...
                        'ub', [0.1; 0; 0.01; 0.01]);
%% Target tube
target_tube = TargetTube('reach-avoid', safe_set, target_set, time_horizon);
```

Problem 1 and 2: Verification and controller synthesis from a given initial state

We will first specify the initial state and parameters for the MATLAB's Global Optimization Toolbox `patternsearch`.

```
initial_state = [-0.75;          % Initial x relative position
                 -0.75;          % Initial y relative position
                 0;              % Initial x relative velocity
                 0];             % Initial y relative velocity
slice_at_vx_vy = initial_state(3:4);
%% Parameters for MATLAB's Global Optimization Toolbox patternsearch
desired_accuracy = 1e-3;         % Decrease for a more accurate lower
                                 % bound at the cost of higher
                                 % computation time
PSoptions = psoptimset('Display','off');
```

Next, using SReachTools, we will compute an optimal open-controller and the associated reach-avoid probability. This function takes about few minutes to run.

```
[lb_stochastic_reach_avoid, optimal_input_vector] = ...
                         getLowerBoundStochReachAvoid(sys,...
                                                      initial_state,...
                                                      target_tube,...
                                                      safe_set,...
                                                      [],...
                                                      desired_accuracy,...
                                                      PSoptions);
fprintf('Maximum reach-avoid probability using an open-loop controller: %1.3f\n', lb_stochasti
```

```
Maximum reach-avoid probability using an open-loop controller: 0.863
```

The function `getLowerBoundStochReachAvoid` uses Fourier transform and convex optimization to underapproximate the reach-avoid problem. Note that `lb_stochastic_reach_avoid` is a lower bound to the maximum attainable reach-avoid probability since using a state-feedback law (also known as a Markov policy) can incorporate more information and attain a higher threshold of safety. Unfortuately, the current state-of-the-art approaches can compute a state-feedback law only using dynamic programming (intractable for a 4D problem) or provide overapproximations of safety (unsuitable for verification).

Using the computed optimal open-loop control law, we can compute the associated optimal mean trajectory.

```
[H_matrix, mean_X_sans_input, ~] =...
       getHmatMeanCovForXSansInput(sys,...
                                   initial_state,...
                                   time_horizon);
optimal_mean_X = mean_X_sans_input + H_matrix * optimal_input_vector;
optimal_mean_trajectory=reshape(optimal_mean_X,sys.state_dimension,[])
```

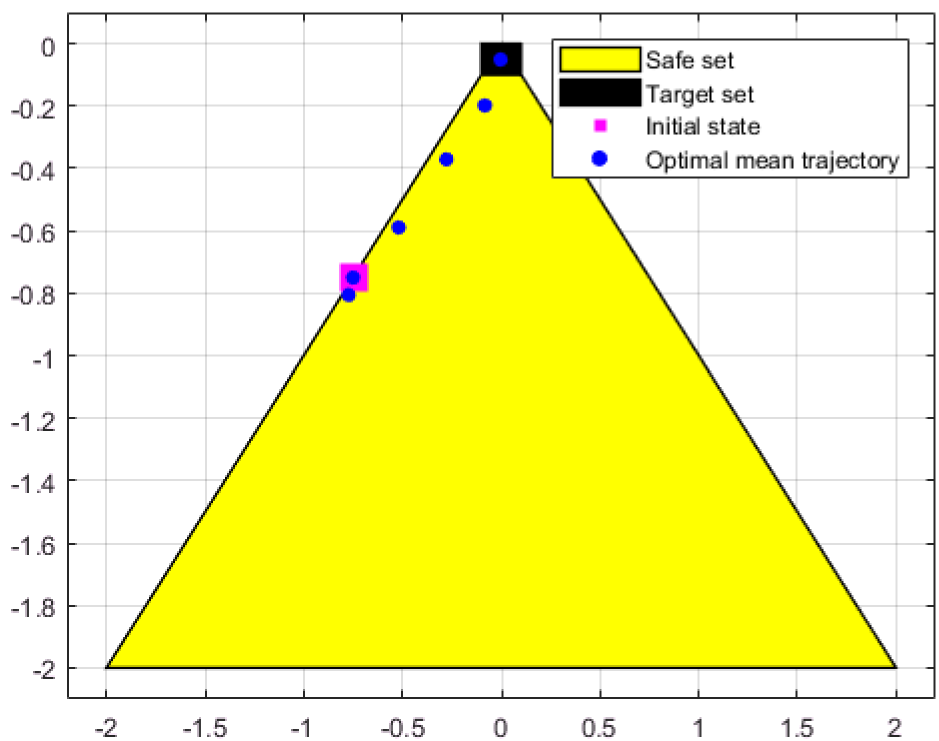
```
optimal_mean_trajectory =
   -0.6942   -0.5432   -0.3527   -0.1775   -0.0112
   -0.7355   -0.6631   -0.5056   -0.2715   -0.0513
    0.0056    0.0095    0.0095    0.0080    0.0086
    0.0014    0.0058    0.0099    0.0135    0.0085
```

•

Visualization of the optimal mean trajectory and the safe and target sets

We can visualize this trajectory along with the specified safe and target sets using MPT3's plot commands.

```
%% Plotting
figure();
box on;
hold on;
plot(safe_set.slice([3,4], slice_at_vx_vy), 'color', 'y');
plot(target_set.slice([3,4], slice_at_vx_vy), 'color', 'k');
scatter(initial_state(1),initial_state(2),200,'ms','filled');
scatter([initial_state(1), optimal_mean_trajectory(1,:)],...
        [initial_state(2), optimal_mean_trajectory(2,:)],...
        30, 'bo', 'filled');
legend_cell = {'Safe set','Target set','Initial state','Optimal mean trajectory'};
legend(legend_cell);
```



Validate the open-loop controller and the obtained lower bound using Monte-Carlo simulations

```
%% Monte-Carlo simulation parameters
n_mcarlo_sims = 1e5;
n_sims_to_plot = 10;
% This function returns the concatenated state vector stacked columnwise
concat_state_realization = generateMonteCarloSims(...
                               n_mcarlo_sims,...
                               sys,...
                               initial_state,...
                               time_horizon,...
```

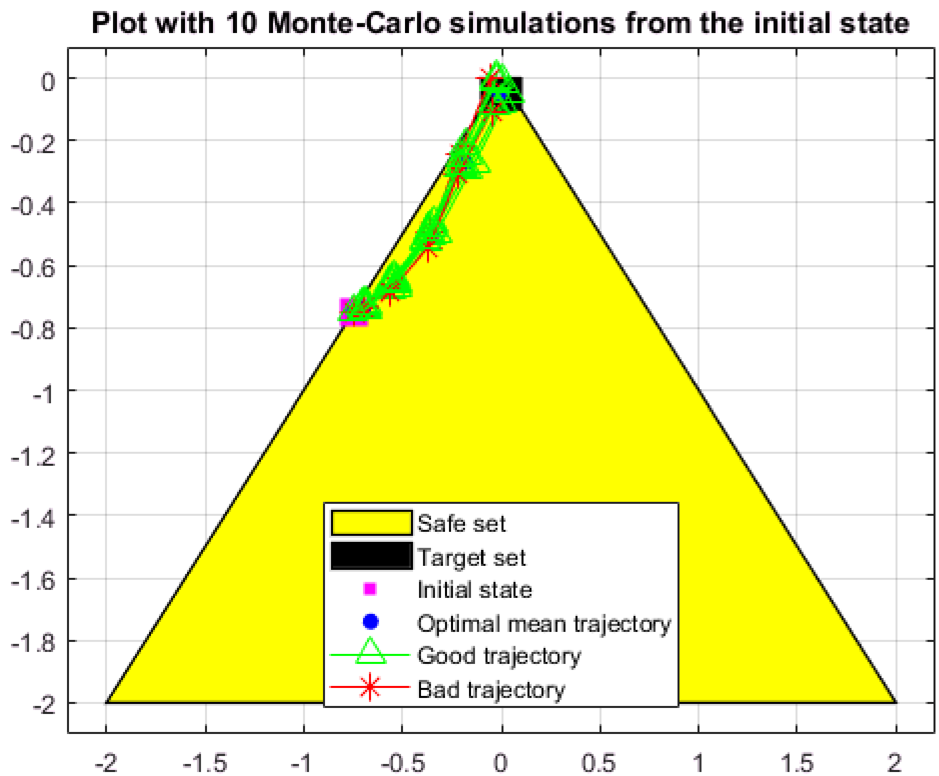
```
                               optimal_input_vector);
% Check if the location is within the target_set or not
mcarlo_result = target_tube.contains(concat_state_realization);
fprintf('Monte-Carlo simulation using %1.0e particles: %1.3f\n',...
        n_mcarlo_sims,...
        sum(mcarlo_result)/n_mcarlo_sims);
```

```
Monte-Carlo simulation using 1e+05 particles: 0.859
```

Plotting random number of trajectories

```
figure();
clf
box on;
hold on;
plot(safe_set.slice([3,4], slice_at_vx_vy), 'color', 'y');
plot(target_set.slice([3,4], slice_at_vx_vy), 'color', 'k');
scatter(initial_state(1),initial_state(2),200,'ms','filled');
scatter([initial_state(1), optimal_mean_trajectory(1,:)],...
        [initial_state(2), optimal_mean_trajectory(2,:)],...
        30, 'bo', 'filled');
legend_cell = {'Safe set','Target set','Initial state','Optimal mean trajectory'};
legend(legend_cell);
%% Plot n_sims_to_plot number of trajectories
green_legend_updated = 0;
red_legend_updated = 0;
traj_indices = floor(n_mcarlo_sims*rand(1,n_sims_to_plot));
for realization_index = traj_indices
    % Check if the trajectory satisfies the reach-avoid objective
    if mcarlo_result(realization_index)
        % Assign green triangle as the marker
        markerString = 'g^-';
    else
        % Assign red asterisk as the marker
        markerString = 'r*-';
    end
    % Create [x(t_1) x(t_2)... x(t_N)]
    reshaped_X_vector = reshape(concat_state_realization(:,realization_index), sys.state_dimer
    % This realization is to be plotted
    h = plot([initial_state(1), reshaped_X_vector(1,:)], ...
             [initial_state(2), reshaped_X_vector(2,:)], ...
             markerString, 'MarkerSize',10);
    % Update the legends if the first else, disable
    if strcmp(markerString,'g^-')
        if green_legend_updated
            h.Annotation.LegendInformation.IconDisplayStyle = 'off';
        else
            green_legend_updated = 1;
            legend_cell{end+1} = 'Good trajectory';
        end
    elseif strcmp(markerString,'r*-')
        if red_legend_updated
            h.Annotation.LegendInformation.IconDisplayStyle = 'off';
        else
            red_legend_updated = 1;
            legend_cell{end+1} = 'Bad trajectory';
        end
    end
```

```
end
legend(legend_cell, 'Location','South');
title(sprintf('Plot with %d Monte-Carlo simulations from the initial state',...
                  n_sims_to_plot));
box on;
grid on;
```



Problem 3: Computation of an underapproximative stochastic reach-avoid set

We will now compute a polytopic underapproximation using the convexity and compactness properties of these sets. Specifically, we can compute the projection of the stochastic reach-avoid set on a 2-dimensional hyperplane on the set of all initial states.

For this example, we consider a hyperplane that fixes the initial velocity. This example sets an initial velocity of $[0.1\ \ 0.1]^\top$. We also specify other parameters needed for this approach. We will reuse the `LtiSystem` object as well as the safe sets and the target sets, `safe_set` and `target_set`.

```
%% Definition of the affine hull
slice_at_vx_vy = ones(2,1)*0.01;                     % The initial velocities of interest
affine_hull_of_interest_2D_A = [zeros(2) eye(2)];
affine_hull_of_interest_2D_b = slice_at_vx_vy;
affine_hull_of_interest_2D = Polyhedron('He',...
                                        [affine_hull_of_interest_2D_A,...
                                         affine_hull_of_interest_2D_b]);
%% Other parameters of the problem
time_horizon=5;
probability_threshold_of_interest = 0.8;     % Stochastic reach-avoid 'level' of interest
no_of_direction_vectors = 8;                 % Increase for a tighter polytopic
                                             % representation at the cost of higher
                                             % computation time
```

```
tolerance_bisection = 1e-2;                      % Tolerance for bisection to compute the
                                                 % extension
%% Parameters for MATLAB's Global Optimization Toolbox patternsearch
desired_accuracy = 1e-3;                         % Decrease for a more accurate lower
                                                 % bound at the cost of higher
                                                 % computation time
PSoptions = psoptimset('Display','off');
```

Construct the polytopic underapproximation of the stochastic reach-avoid set. The function `getFtUnderapproxStochReachAvoidSet` will provide the polytope (n-dimensional) and the optimal open-loop controllers for each of the vertices, along with other useful information. This function will take ~ 20 minutes to run. The vertices are computed by performing bisection along a set of direction vectors originating from a point that is guaranteed to be in the polytope. We choose the guaranteed point to be the initial state that has the maximum probability of success, refered to as `xmax`. Computation of `xmax` is a concave maximization problem.

```
[underapproximate_stochastic_reach_avoid_polytope,...
 optimal_input_vector_at_boundary_points,...
 xmax,...
 optimal_input_vector_for_xmax,...
 maximum_underapproximate_reach_avoid_probability,...
 optimal_theta_i,...
 optimal_reachAvoid_i] =...
          getUnderapproxStochReachAvoidSet(sys,...
                                           target_tube,...
                                           safe_set, ...
                                           probability_threshold_of_interest,...
                                           tolerance_bisection,...
                                           no_of_direction_vectors,...
                                           affine_hull_of_interest_2D,...
                                           desired_accuracy,...
                                           PSoptions);
```

```
Computing the x_max for the Fourier transform-based underapproximation
Polytopic underapproximation exists for alpha = 0.80 since W(x_max) = 0.866.

Analyzing direction (shown transposed) :1/8
    -1     0     0     0

Upper bound of theta: 0.43
OptRAProb | OptTheta | LB_theta | UB_theta | OptInp^2 | Exit reason
  0.8540  |  0.2137  |  0.0000  |  0.4273  |  0.0188  | Feasible
  0.8660  |  0.3205  |  0.2137  |  0.4273  |  0.0171  | Feasible
  0.8660  |  0.3739  |  0.3205  |  0.4273  |  0.0201  | Feasible
  0.8660  |  0.4006  |  0.3739  |  0.4273  |  0.0199  | Feasible
  0.8610  |  0.4140  |  0.4006  |  0.4273  |  0.0221  | Feasible
  0.8580  |  0.4207  |  0.4140  |  0.4273  |  0.0265  | Feasible
Analyzing direction (shown transposed) :2/8
   -0.7071   -0.7071         0         0

Upper bound of theta: 1.39
OptRAProb | OptTheta | LB_theta | UB_theta | OptInp^2 | Exit reason
  0.8660  |  0.0000  |  0.0000  |  1.3933  |  0.0108  | Infeasible
  0.8660  |  0.0000  |  0.0000  |  0.6966  |  0.0108  | Infeasible
  0.8660  |  0.1742  |  0.0000  |  0.3483  |  0.0202  | Feasible
  0.8660  |  0.2612  |  0.1742  |  0.3483  |  0.0283  | Feasible
  0.8660  |  0.3048  |  0.2612  |  0.3483  |  0.0303  | Feasible
  0.8660  |  0.3265  |  0.3048  |  0.3483  |  0.0240  | Feasible
  0.8660  |  0.3374  |  0.3265  |  0.3483  |  0.0236  | Feasible
  0.8650  |  0.3429  |  0.3374  |  0.3483  |  0.0235  | Feasible
Analyzing direction (shown transposed) :3/8
```

```
   -0.0000   -1.0000         0         0

Upper bound of theta: 0.99
OptRAProb | OptTheta | LB_theta | UB_theta | OptInp^2 | Exit reason
  0.8660  |  0.0000  |  0.0000  |  0.9852  |  0.0108  | Infeasible
  0.8660  |  0.2463  |  0.0000  |  0.4926  |  0.0092  | Feasible
  0.8660  |  0.3694  |  0.2463  |  0.4926  |  0.0277  | Feasible
  0.8660  |  0.4310  |  0.3694  |  0.4926  |  0.0333  | Feasible
  0.8660  |  0.4618  |  0.4310  |  0.4926  |  0.0330  | Feasible
  0.8650  |  0.4772  |  0.4618  |  0.4926  |  0.0361  | Feasible
  0.8660  |  0.4849  |  0.4772  |  0.4926  |  0.0377  | Feasible
Analyzing direction (shown transposed) :4/8
    0.7071   -0.7071         0         0

Upper bound of theta: 1.39
OptRAProb | OptTheta | LB_theta | UB_theta | OptInp^2 | Exit reason
  0.8660  |  0.0000  |  0.0000  |  1.3933  |  0.0108  | Infeasible
  0.8660  |  0.0000  |  0.0000  |  0.6966  |  0.0108  | Infeasible
  0.8660  |  0.1742  |  0.0000  |  0.3483  |  0.0173  | Feasible
  0.8660  |  0.2612  |  0.1742  |  0.3483  |  0.0300  | Feasible
  0.8660  |  0.3048  |  0.2612  |  0.3483  |  0.0364  | Feasible
  0.8660  |  0.3265  |  0.3048  |  0.3483  |  0.0323  | Feasible
  0.8660  |  0.3374  |  0.3265  |  0.3483  |  0.0306  | Feasible
  0.8660  |  0.3429  |  0.3374  |  0.3483  |  0.0306  | Feasible
Analyzing direction (shown transposed) :5/8
    1.0000   -0.0000         0         0

Upper bound of theta: 1.60
OptRAProb | OptTheta | LB_theta | UB_theta | OptInp^2 | Exit reason
  0.8660  |  0.0000  |  0.0000  |  1.6023  |  0.0108  | Infeasible
  0.8660  |  0.0000  |  0.0000  |  0.8011  |  0.0108  | Infeasible
  0.8660  |  0.2003  |  0.0000  |  0.4006  |  0.0157  | Feasible
  0.8650  |  0.3004  |  0.2003  |  0.4006  |  0.0185  | Feasible
  0.8660  |  0.3505  |  0.3004  |  0.4006  |  0.0188  | Feasible
  0.8660  |  0.3755  |  0.3505  |  0.4006  |  0.0195  | Feasible
  0.8660  |  0.3880  |  0.3755  |  0.4006  |  0.0195  | Feasible
  0.8660  |  0.3943  |  0.3880  |  0.4006  |  0.0201  | Feasible
Analyzing direction (shown transposed) :6/8
    0.7071    0.7071         0         0

Upper bound of theta: 1.13
OptRAProb | OptTheta | LB_theta | UB_theta | OptInp^2 | Exit reason
  0.8660  |  0.0000  |  0.0000  |  1.1330  |  0.0108  | Infeasible
  0.8660  |  0.0000  |  0.0000  |  0.5665  |  0.0108  | Infeasible
  0.8660  |  0.1416  |  0.0000  |  0.2832  |  0.0203  | Feasible
  0.8660  |  0.2124  |  0.1416  |  0.2832  |  0.0209  | Feasible
  0.8660  |  0.2478  |  0.2124  |  0.2832  |  0.0170  | Feasible
  0.8660  |  0.2655  |  0.2478  |  0.2832  |  0.0170  | Feasible
  0.8660  |  0.2744  |  0.2655  |  0.2832  |  0.0177  | Feasible
Analyzing direction (shown transposed) :7/8
    0.0000    1.0000         0         0

Upper bound of theta: 0.43
OptRAProb | OptTheta | LB_theta | UB_theta | OptInp^2 | Exit reason
  0.8640  |  0.2137  |  0.0000  |  0.4273  |  0.0184  | Feasible
  0.8660  |  0.3205  |  0.2137  |  0.4273  |  0.0236  | Feasible
  0.8630  |  0.3739  |  0.3205  |  0.4273  |  0.0228  | Feasible
  0.8580  |  0.4006  |  0.3739  |  0.4273  |  0.0201  | Feasible
  0.8560  |  0.4140  |  0.4006  |  0.4273  |  0.0151  | Feasible
  0.8440  |  0.4207  |  0.4140  |  0.4273  |  0.0171  | Feasible
Analyzing direction (shown transposed) :8/8
   -0.7071    0.7071         0         0

Upper bound of theta: 0.30
OptRAProb | OptTheta | LB_theta | UB_theta | OptInp^2 | Exit reason
  0.8660  |  0.1511  |  0.0000  |  0.3022  |  0.0138  | Feasible
  0.8650  |  0.2266  |  0.1511  |  0.3022  |  0.0274  | Feasible
  0.8660  |  0.2644  |  0.2266  |  0.3022  |  0.0310  | Feasible
```

```
  0.8640  |  0.2833  |  0.2644  |  0.3022  |  0.0328  | Feasible
  0.8520  |  0.2927  |  0.2833  |  0.3022  |  0.0333  | Feasible
```

While the open-loop controllers are available only for the vertices, the convexity of the computged underapproximation suggests that a convex combination of the open-loop controllers can be a good initial guess for any point within the underapproximative polytope.

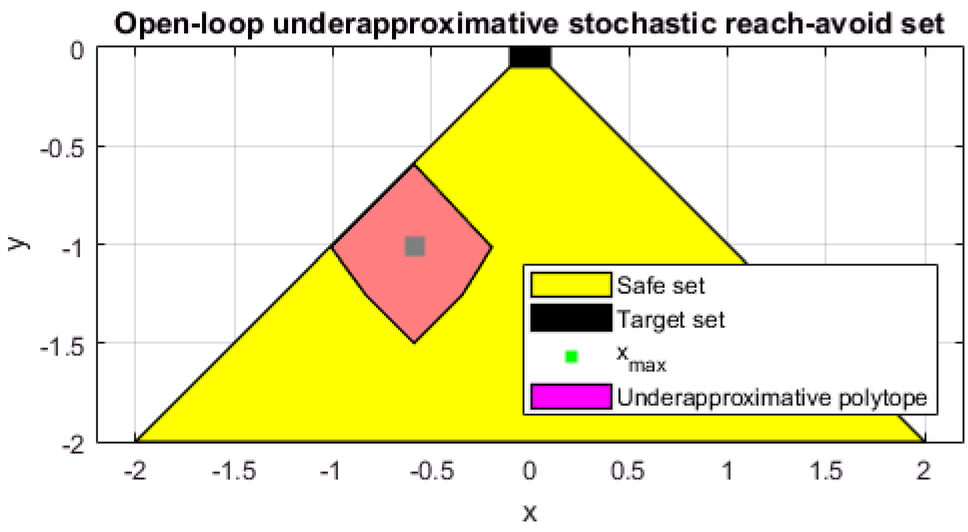
Visualization of the underapproximative polytope and the safe and target sets

Construct the 2D representation of the underapproximative polytope.

```
set_of_direction_vectors = computeDirectionVectors(no_of_direction_vectors,...
                                                   sys.state_dimension,...
                                                   affine_hull_of_interest_2D);
vertex_poly = xmax + optimal_theta_i.* set_of_direction_vectors;
underapproximate_stochastic_reach_avoid_polytope_2D =...
                                            Polyhedron('V',vertex_poly(1:2,:)');
```

Plot the underapproximative polytope along with the safe and the target sets.

```
figure();
hold on;
plot(safe_set.slice([3,4], slice_at_vx_vy), 'color', 'y');
plot(target_set.slice([3,4], slice_at_vx_vy), 'color', 'k');

scatter(xmax(1), xmax(2), 100,'gs','filled')
if ~isEmptySet(underapproximate_stochastic_reach_avoid_polytope)
    plot(underapproximate_stochastic_reach_avoid_polytope_2D,...
         'color','m','alpha',0.5);
    leg=legend({'Safe set',...
            'Target set',...
            'x_{max}',...
            'Underapproximative polytope'});
else
    leg=legend({'Safe set','Target set', 'x_{max}'})
end
set(leg,'Location','SouthEast');
xlabel('x')
ylabel('y')
axis equal
box on;
grid on;
title('Open-loop underapproximative stochastic reach-avoid set');
```



Validate the underapproximative set and the controllers synthesized using Monte-Carlo simulations

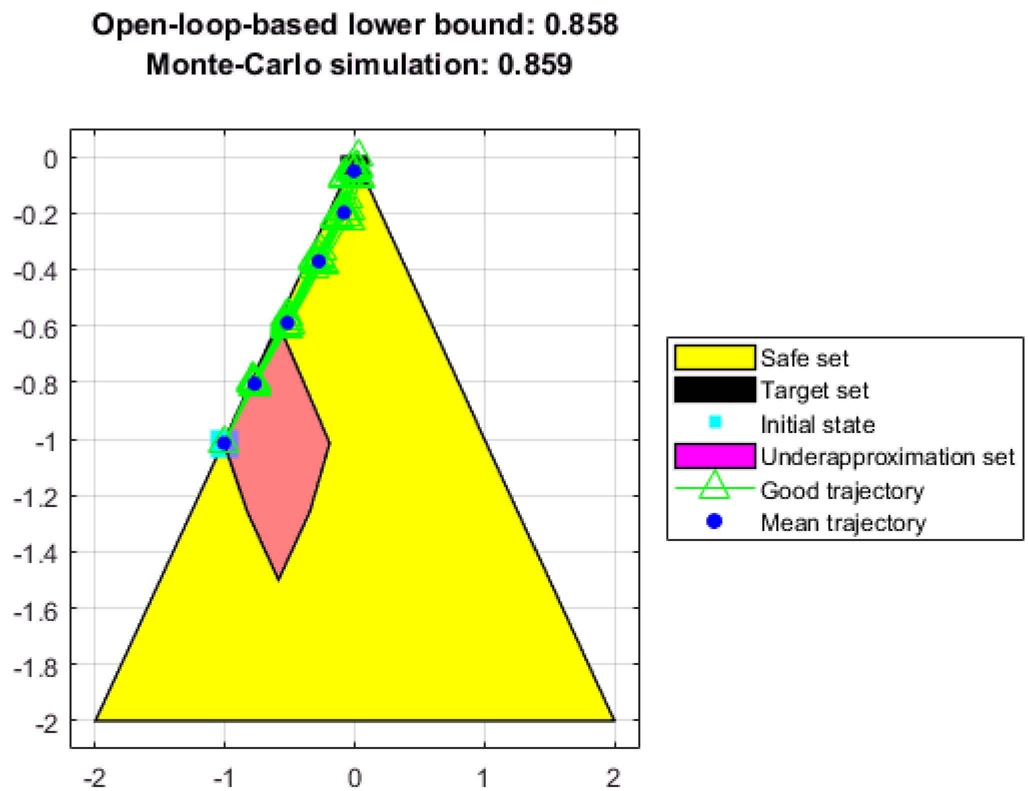
We will now check how the optimal policy computed for each corners perform in Monte-Carlo simulations.

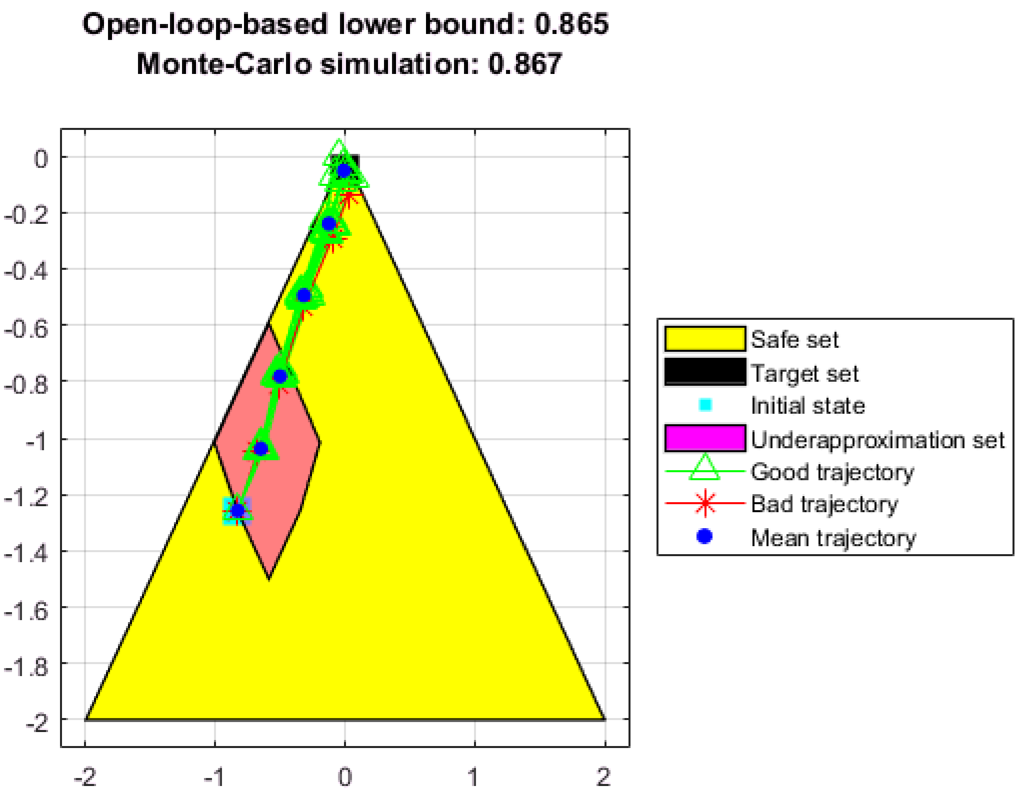
```
if ~isEmptySet(underapproximate_stochastic_reach_avoid_polytope)
    for direction_index = 1:no_of_direction_vectors
        figure();
        hold on;
        plot(safe_set.slice([3,4], slice_at_vx_vy), 'color', 'y');
        plot(target_set.slice([3,4], slice_at_vx_vy), 'color', 'k');
        scatter(vertex_poly(1,direction_index),...
                vertex_poly(2,direction_index),...
                200,'cs','filled');
        plot(underapproximate_stochastic_reach_avoid_polytope_2D,...
             'color','m','alpha',0.5);
        legend_cell = {'Safe set',...
                       'Target set',...
                       'Initial state',...
                       'Underapproximation set'};
        concat_state_realization = generateMonteCarloSims(...
                                       n_mcarlo_sims,...
                                       sys,...
                                       vertex_poly(:,direction_index),...
                                       time_horizon,...
                                       optimal_input_vector_at_boundary_points(:,direction_inc
        % Check if the location is within the target_set or not
        mcarlo_result = target_tube.contains(concat_state_realization);
        %% Plot n_sims_to_plot number of trajectories
```

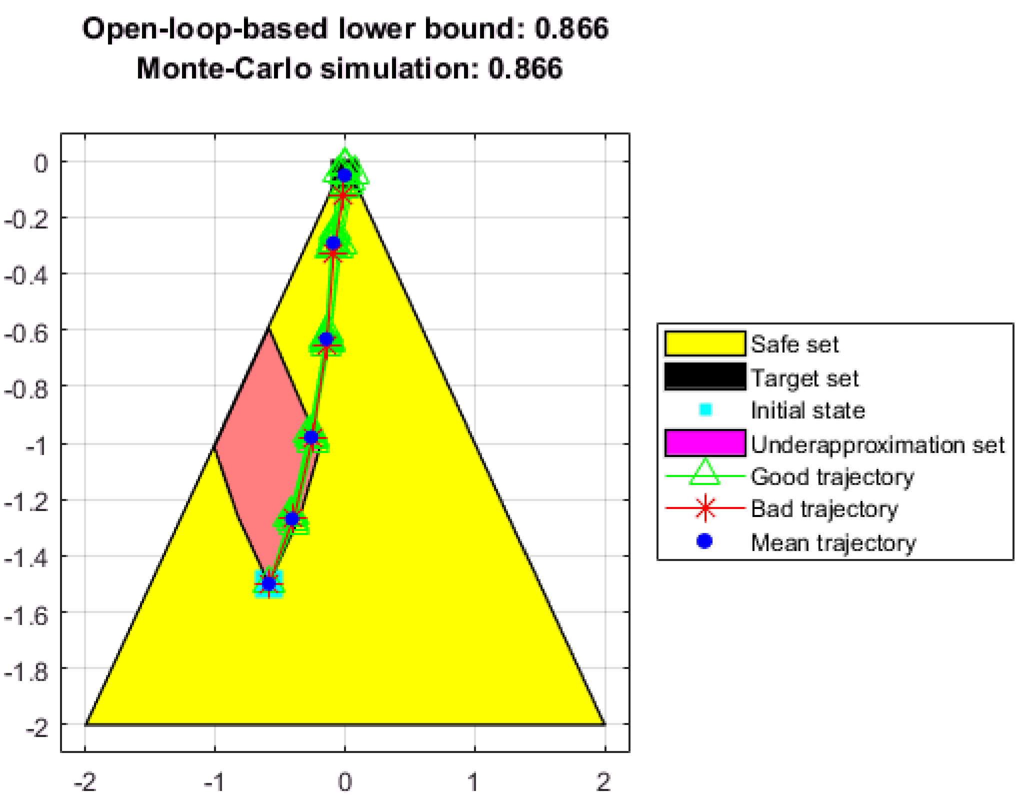
```
        green_legend_updated = 0;
        red_legend_updated = 0;
        traj_indices = floor(n_mcarlo_sims*rand(1,n_sims_to_plot));
        for realization_index = traj_indices
            % Check if the trajectory satisfies the reach-avoid objective
            if mcarlo_result(realization_index)
                % Assign green triangle as the marker
                markerString = 'g^-';
            else
                % Assign red asterisk as the marker
                markerString = 'r*-';
            end
            % Create [x(t_1) x(t_2)... x(t_N)]
            reshaped_X_vector = reshape(concat_state_realization(:,realization_index), sys.sta
            % This realization is to be plotted
            h = plot([vertex_poly(1,direction_index), reshaped_X_vector(1,:)], ...
                     [vertex_poly(2,direction_index), reshaped_X_vector(2,:)], ...
                     markerString, 'MarkerSize',10);
            % Update the legends if the first else, disable
            if strcmp(markerString,'g^-')
                if green_legend_updated
                    h.Annotation.LegendInformation.IconDisplayStyle = 'off';
                else
                    green_legend_updated = 1;
                    legend_cell{end+1} = 'Good trajectory';
                end
            elseif strcmp(markerString,'r*-')
                if red_legend_updated
                    h.Annotation.LegendInformation.IconDisplayStyle = 'off';
                else
                    red_legend_updated = 1;
                    legend_cell{end+1} = 'Bad trajectory';
                end
            end
        end
        % Compute and plot the mean trajectory under the optimal open-loop
        % controller from the the vertex under study
        [H_matrix, mean_X_sans_input, ~] =...
         getHmatMeanCovForXSansInput(sys,...
                                     vertex_poly(:,direction_index),...
                                     time_horizon);
        optimal_mean_X = mean_X_sans_input + H_matrix *...
                    optimal_input_vector_at_boundary_points(:, direction_index);
        optimal_mean_trajectory=reshape(optimal_mean_X,sys.state_dimension,[]);
        % Plot the optimal mean trajectory from the vertex under study
        scatter(...
              [vertex_poly(1,direction_index), optimal_mean_trajectory(1,:)],...
              [vertex_poly(2,direction_index), optimal_mean_trajectory(2,:)],...
              30, 'bo', 'filled');
        legend_cell{end+1} = 'Mean trajectory';
        leg = legend(legend_cell,'Location','EastOutside');
        % title for the plot
        title(sprintf(['Open-loop-based lower bound: %1.3f\n Monte-Carlo ',...
                       'simulation: %1.3f\n'],...
                optimal_reachAvoid_i(direction_index),...
                sum(mcarlo_result)/n_mcarlo_sims));
        box on;
        grid on;

        fprintf(['Open-loop-based lower bound and Monte-Carlo simulation ',...
                 '(%1.0e particles): %1.3f, %1.3f\n'],...
```

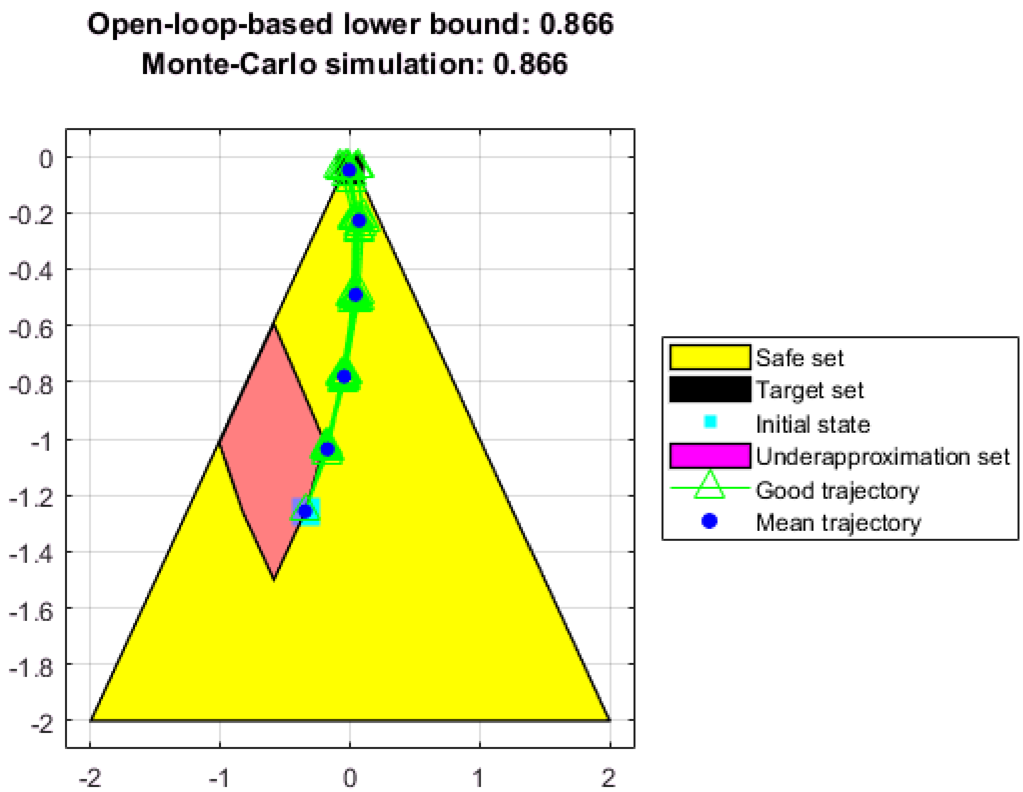
```
                n_mcarlo_sims,...
                optimal_reachAvoid_i(direction_index),...
                sum(mcarlo_result)/n_mcarlo_sims);
    end
end
```

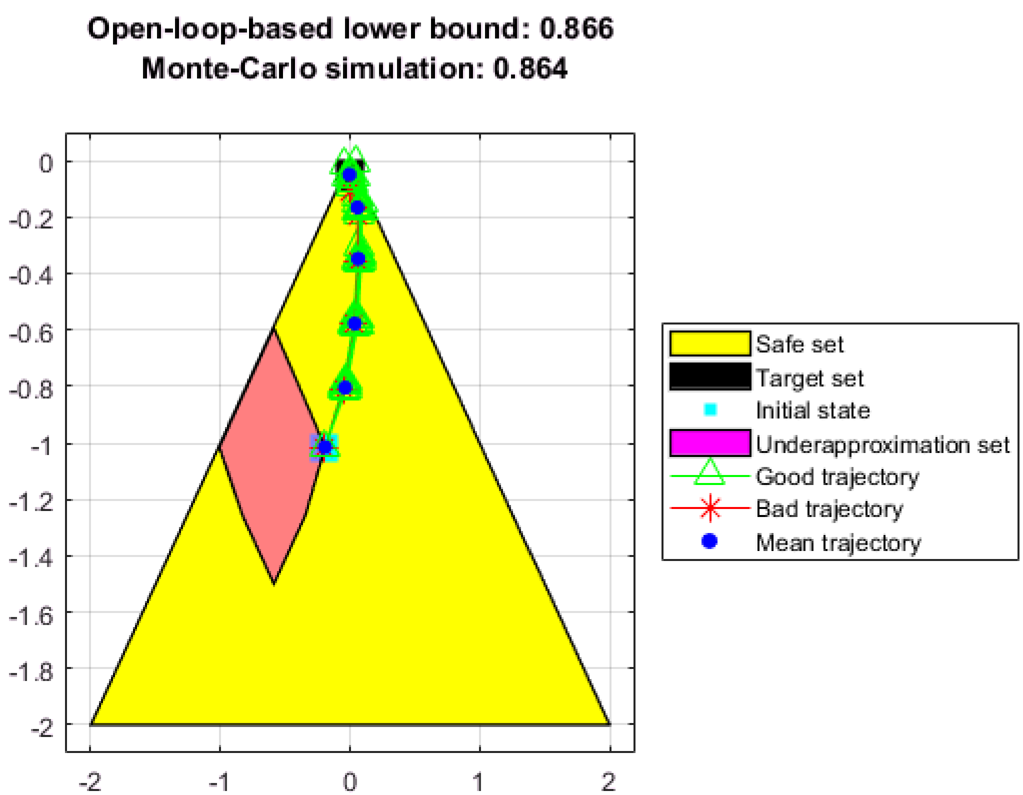
```
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.858, 0.859
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.865, 0.867
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.866, 0.866
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.866, 0.866
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.866, 0.864
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.866, 0.861
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.844, 0.844
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.852, 0.841
```

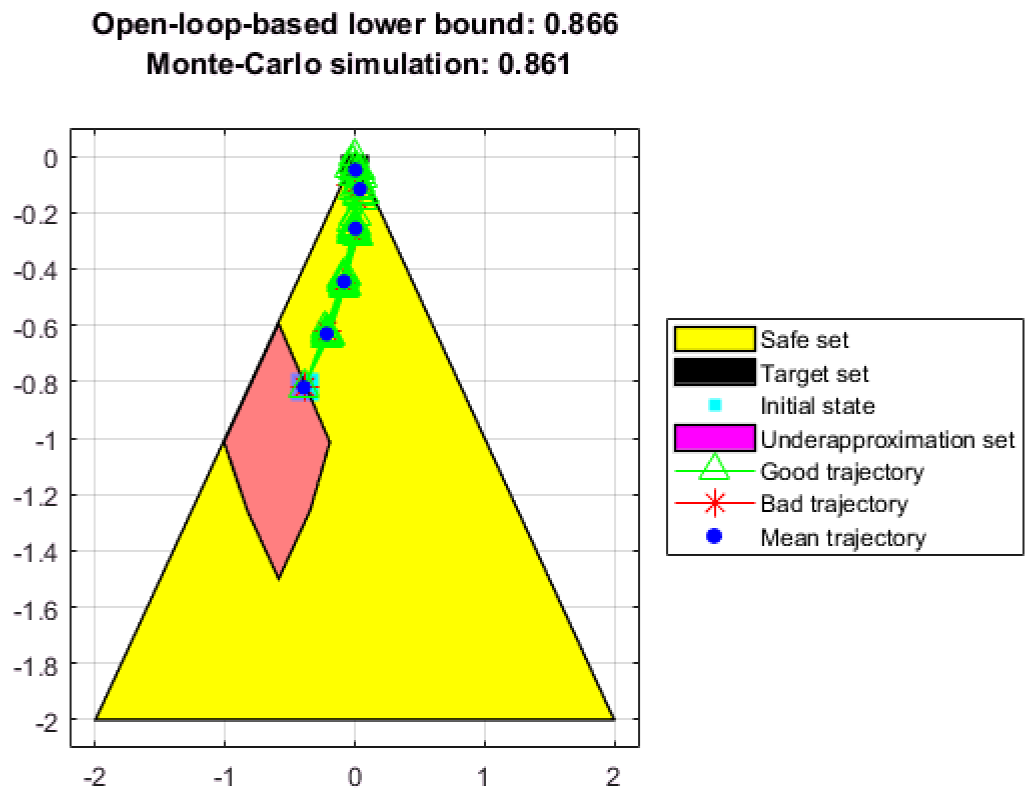


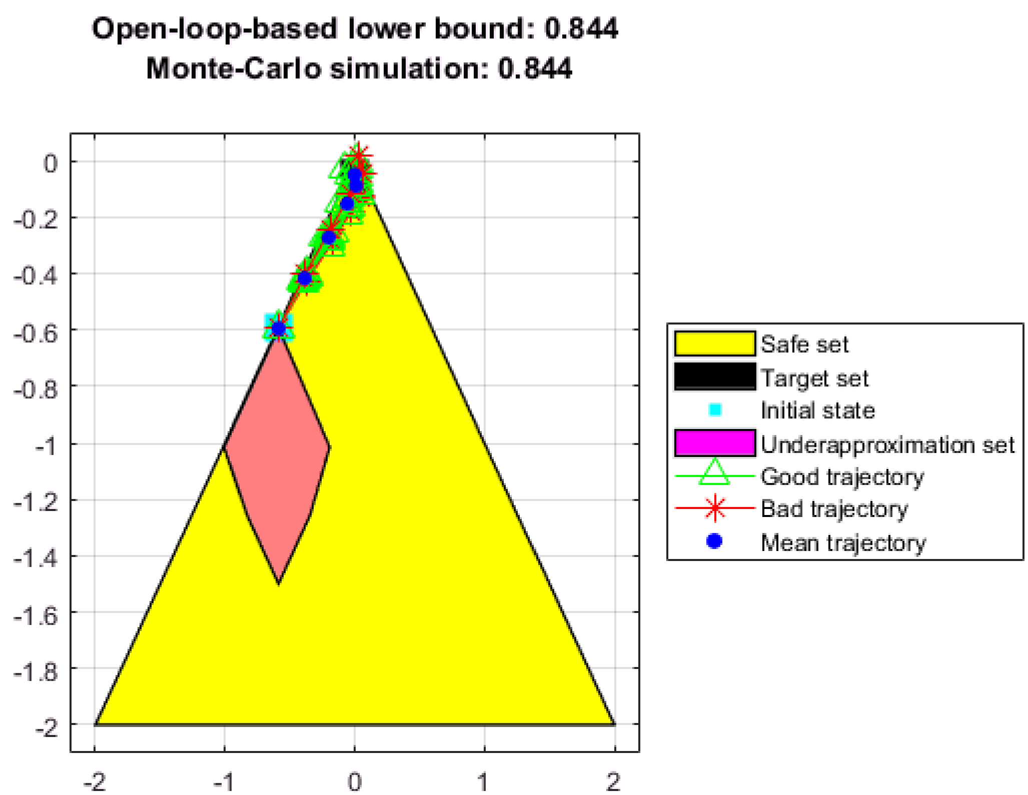

Open-loop-based lower bound: 0.865
Monte-Carlo simulation: 0.867
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
-2
-2
-1
0
1
2
Safe set
Target set
Initial state
Underapproximation set
Good trajectory
Bad trajectory
Mean trajectory

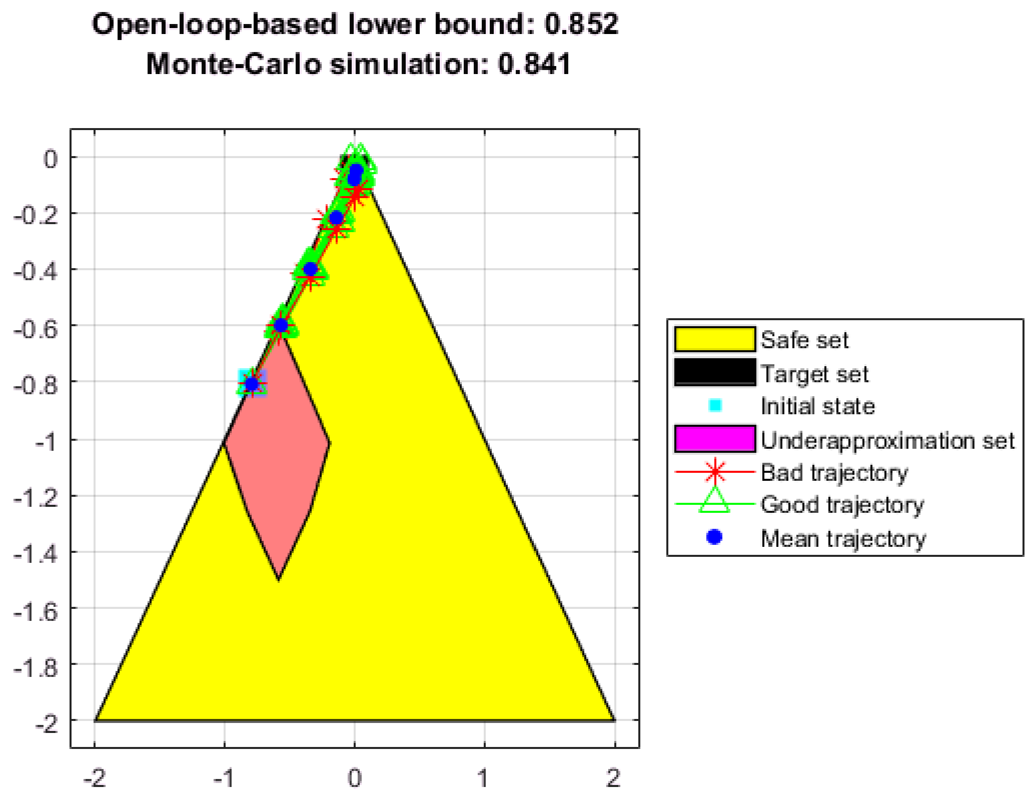

Open-loop-based lower bound: 0.866
Monte-Carlo simulation: 0.866
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
-2
-2
-1
0
1
2
Safe set
Target set
Initial state
Underapproximation set
Good trajectory
Bad trajectory
Mean trajectory


Open-loop-based lower bound: 0.866
Monte-Carlo simulation: 0.866
Safe set
Target set
Initial state
Underapproximation set
Good trajectory
Mean trajectory


Open-loop-based lower bound: 0.866
Monte-Carlo simulation: 0.864
Safe set
Target set
Initial state
Underapproximation set
Good trajectory
Bad trajectory
Mean trajectory


Open-loop-based lower bound: 0.866
Monte-Carlo simulation: 0.861
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
-2
-2
-1
0
1
2
Safe set
Target set
Initial state
Underapproximation set
Good trajectory
Bad trajectory
Mean trajectory


Open-loop-based lower bound: 0.844
Monte-Carlo simulation: 0.844
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
-2
-2
-1
0
1
2
Safe set
Target set
Initial state
Underapproximation set
Good trajectory
Bad trajectory
Mean trajectory


Open-loop-based lower bound: 0.852
Monte-Carlo simulation: 0.841
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
-2
-2
-1
0
1
2
Safe set
Target set
Initial state
Underapproximation set
Bad trajectory
Good trajectory
Mean trajectory